Bash Shell and Commands

Introduction

What is Shell?

"Shell" is the UNIX term for a user interface to the system—something that lets you communicate with the computer via the keyboard and the display. Shells are just separate programs that encapsulate the system, and, as such, there are many to choose from.

Systems are usually set up with a "standard" shell but there are many useful versions available.

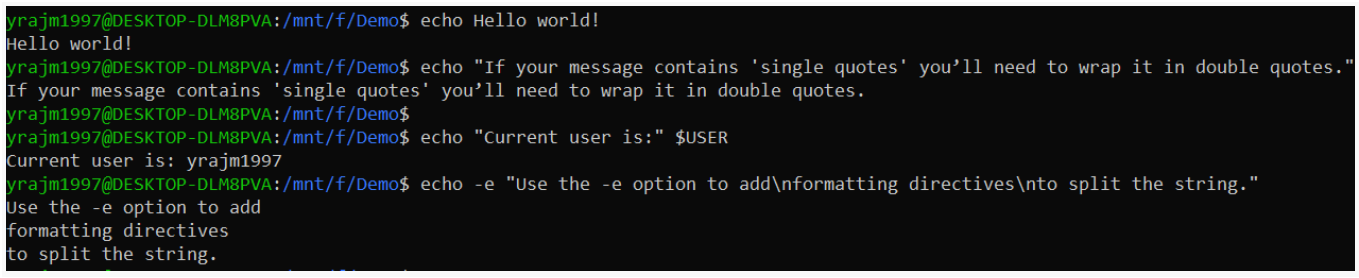
bash or Bourne Again SHell is a modern general purpose shell.

Simple Bash Commands

How to print text - echo

For printing messages, use the echo command. It will also allow printing variables and text formatting. Check the examples below:

```
echo Hello world!
echo "If your message contains 'single quotes' you’ll need to wrap it in double quotes."
echo "Current user is:" $USER
echo -e "Use the -e option to add\nformatting directives\nto split the string."
```

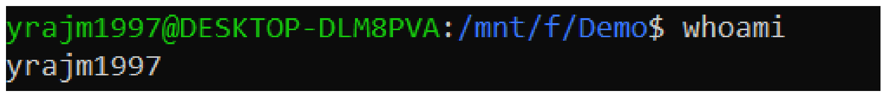


```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ echo Hello world!
Hello world!
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ echo "If your message contains 'single quotes' you’ll need to wrap it in double quotes."
If your message contains 'single quotes' you’ll need to wrap it in double quotes.
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ echo "Current user is:" $USER
Current user is: yrajm1997
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ echo -e "Use the -e option to add\nformatting directives\nto split the string."
Use the -e option to add
formatting directives
to split the string.
```

whoami

To print the user name currently logged in to the terminal session

```
whoami
```

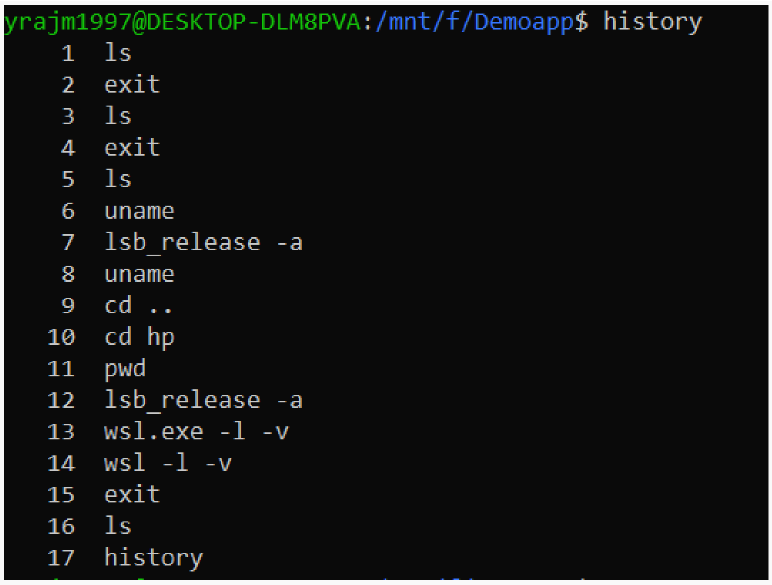


```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ whoami
yrajm1997
```

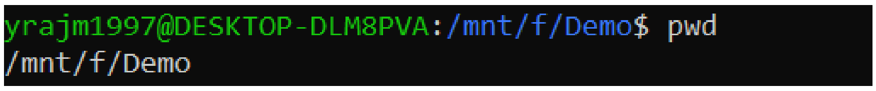
history

The history command displays a list (i.e., the history) of commands that you executed in the current command shell, as shown here:

```
history
```


```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demoapp$ history
    1  ls
    2  exit
    3  ls
    4  exit
    5  ls
    6  uname
    7  lsb_release -a
    8  uname
    9  cd ..
   10  cd hp
   11  pwd
   12  lsb_release -a
   13  wsl.exe -l -v
   14  wsl -l -v
   15  exit
   16  ls
   17  history
```

pwd
A frequently used Bash command is pwd ("print working directory") that displays the current directory, as shown here:
`pwd`


```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ pwd
/mnt/f/Demo
```

Concept of root and home directory
The Linux **home directory** is a directory for a particular user of the system and consists of individual files. It is also referred to as the login directory. This is the first place that occurs after logging into a Linux system. It is automatically created as "**/home**" for each user in the directory'. It is a standard subdirectory of the root directory.
The **root directory** contains all other directories, subdirectories, and files on the system. It is denoted by a forward slash (**/**).

The home directory can be said as a personal working space for all the users except root. There is a separate directory for every user.

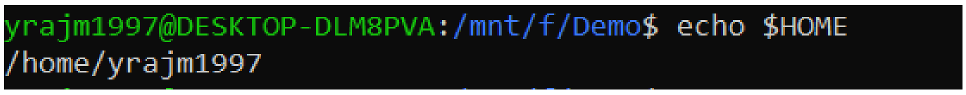
To see a full path to your home directory execute the following command:

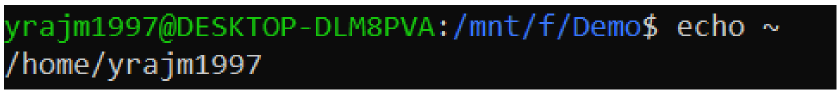
`echo $HOME`

OR

`echo ~`

The tilde (~) represents the home directory of the currently logged in user.


```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ echo $HOME
/home/yrajm1997
```


```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ echo ~
/home/yrajm1997
```

Setting PATH environment variable

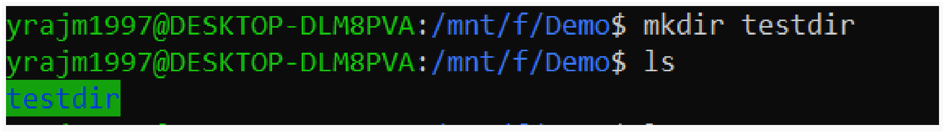
If a directory containing an executable file is not included in your PATH environment variable, simply add that directory to your PATH environment variable so that you can invoke an executable file by specifying just the filename:

```
export PATH=$HOME/anaconda:$PATH
```

mkdir

To create a new directory

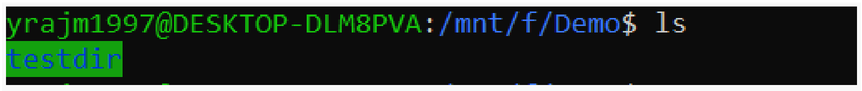
```
mkdir testdir
```



```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ mkdir testdir
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ ls
testdir
```

ls

To list all the files that the folder contains

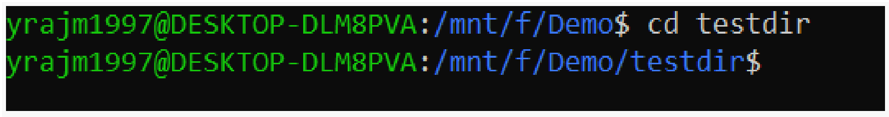


```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ ls
testdir
```

cd

Once you have a folder, you can move into it using the *cd (change directory)* command. You invoke it specifying a folder to move into. You can specify a folder name, or an entire path.

```
cd testdir
```

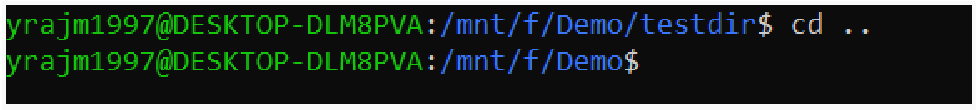


```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ cd testdir
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo/testdir$
```

Back to the parent folder - *concept of . (dot) and .. (dot dot)*

You can use the **..** special path to indicate the parent folder:

```
cd ..
```



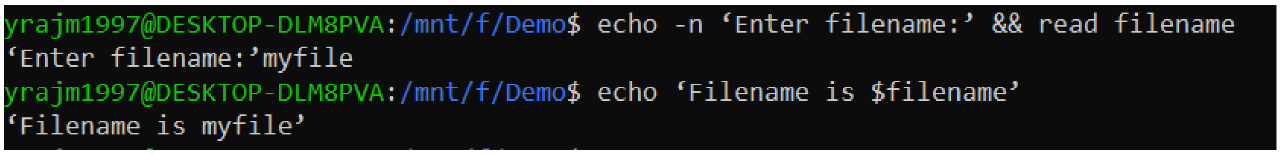
```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo/testdir$ cd ..
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$
```

There is another special path indicator which is **.** , and indicates the current folder.

Reading User Input

read command can be used to get user input into shell variables. We can use echo to prompt the user, like this:

```
echo -n ‘Enter filename:’ && read filename
echo ‘Filename is $filename’
```



```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ echo -n ‘Enter filename:’ && read filename
‘Enter filename:’myfile
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ echo ‘Filename is $filename’
‘Filename is myfile’
```

Working with FIles

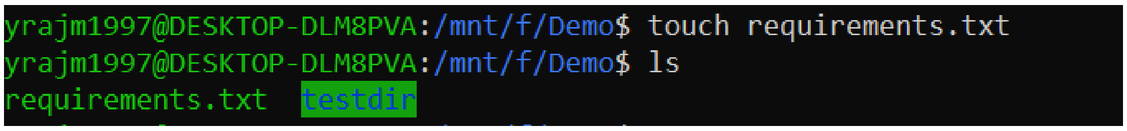
Create files

Create an empty file using the touch command:

```
touch <filename.extension>
```

For example:

```
touch requirements.txt
```



```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ touch requirements.txt
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ ls
requirements.txt  testdir
```

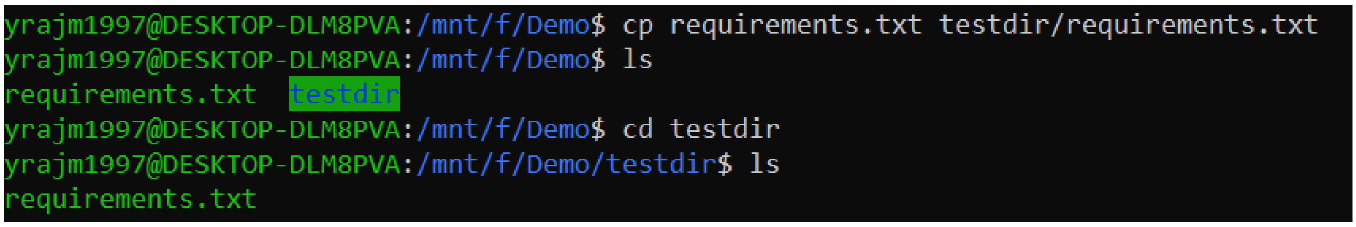
Copy files

Copy the file to another location:

```
cp <source file path> <destination path>
```

For example:

```
cp requirements.txt testdir/requirements.txt
```



```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ cp requirements.txt testdir/requirements.txt
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ ls
requirements.txt  testdir
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ cd testdir
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo/testdir$ ls
requirements.txt
```

Note: If the destination path is a file, the contents of the source file will overwrite the contents of the destination file.

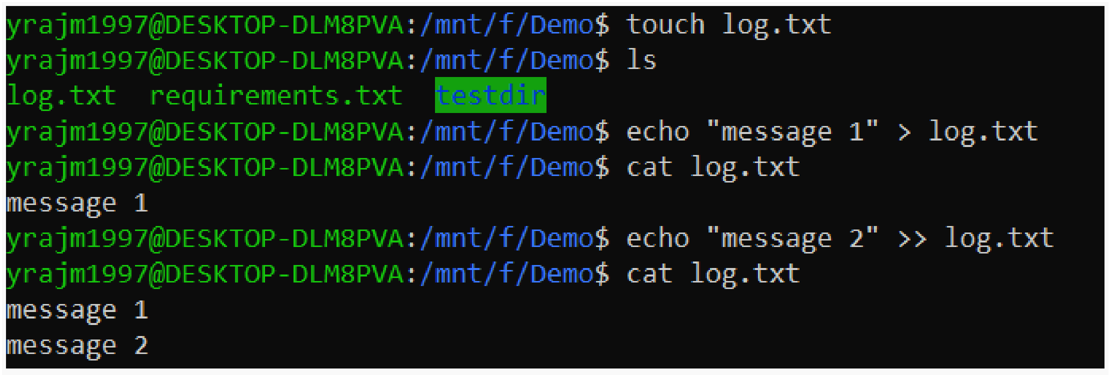
Recursively copy all files from one folder to another by using the -r switch:

```
cp -r <source directory path> <destination directory path>
```

Write to file

Use ‘>’ to write into a file and ‘>>’ to append to a file

```
touch log.txt                     # Creates a new file
echo "message 1" > log.txt        # Adds message 1 line to file
echo "message 2" >> log.txt         # Appends message 2 after message 1
```


```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ touch log.txt
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ ls
log.txt  requirements.txt  testdir
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ echo "message 1" > log.txt
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ cat log.txt
message 1
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ echo "message 2" >> log.txt
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ cat log.txt
message 1
message 2
```

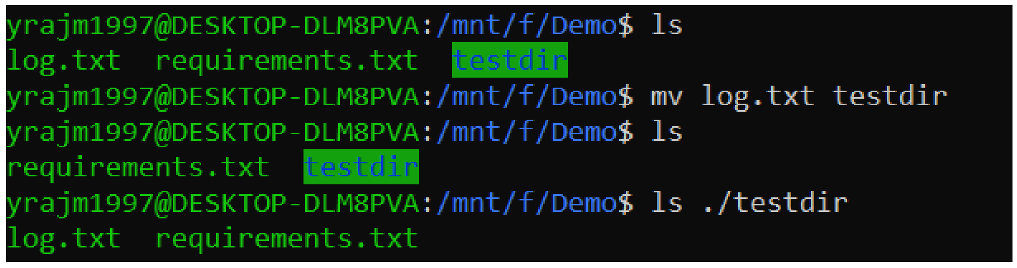
Move and Rename files

The *mv* command is equivalent to a combination of *cp* and *rm*. You can use this command to move multiple files to a directory:

```
mv <file> <destination directory>
mv <directory> <destination directory>
```

For example:

```
mv log.txt testdir
```

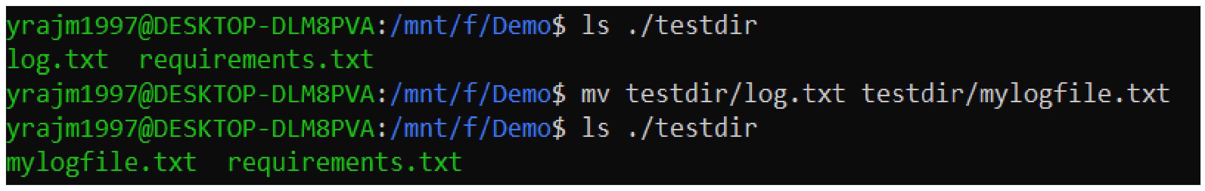

```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ ls
log.txt  requirements.txt  testdir
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ mv log.txt testdir
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ ls
requirements.txt  testdir
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ ls ./testdir
log.txt  requirements.txt
```

To rename a file or directory, simply move it to its current location and specify a new name:

```
mv <filename> <new filename>
mv <directory> <new directory name>
```

For example:

```
mv testdir/log.txt testdir/mylogfile.txt
```


```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ ls ./testdir
log.txt  requirements.txt
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ mv testdir/log.txt testdir/mylogfile.txt
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ ls ./testdir
mylogfile.txt  requirements.txt
```

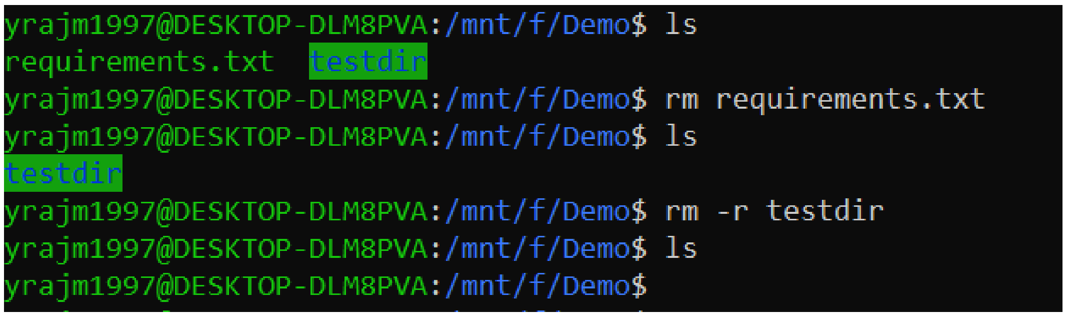
Remove files

The *rm* command removes files and when you specify the *-r* option, the *rm* command removes the contents of a directory (as well as the directory).

```
rm <filename>
rm -r <directory>
```

For example:

```
rm requirements.txt
rm -r testdir
```



```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ ls
requirements.txt  testdir
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ rm requirements.txt
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ ls
testdir
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ rm -r testdir
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ ls
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$
```

To remove all the files and subdirectories of a folder, you can use -r (recursive) and -f (force) options:

```
rm -rf <directory>
```

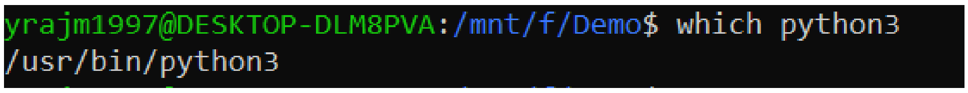
which

Suppose you have a command you can execute, because it's in the shell path, but you want to know where it is located. You can do so using *which*. The command will return the path to the command specified:

```
which <executable>
```

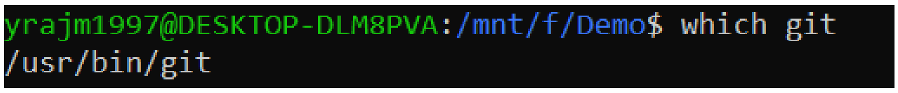
For example:

```
which python3
```



```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ which python3
/usr/bin/python3
```

```
which git
```



```
yrajm1997@DESKTOP-DLM8PVA:/mnt/f/Demo$ which git
/usr/bin/git
```

Note that the *which* command will only work for executables stored on disk, not aliases or built-in shell functions.